Fermi National Accelerator Laboratory

TM-1433
2603.001

Image Digitizer System for Bubble Chamber Laser

Herman Haggerty

December 8, 1986

Operated by Universities Research Association Inc. under contract with the United States Department of Energy

December 8, 1986

Image Digitizer System for Bubble Chamber Laser

Herman Haggerty
Research Facilities Department

An IBM PC based image digitizer system has been assembled to monitor the laser flash used for holography at the 15 foot bubble chamber. The hardware and the operating software will be outlined here.

A CCD black and white camera (Sony XC-38) and its companion power divider (Sony DC-38) are connected to a separate D.C. power source (12V 0.3A). The camera generates its own sync and outputs a composite video through a single RG 174 cable. The camera inputs directly to spigot #2 of an image processing board (Matrox PIP-512) located in an expansion slot of an IBM PC/XT. This board digitizes the video signal and then outputs directly to a RBG monitor (Sony CDP 1201). In another expansion slot is a digital input card (ICS DIM 32) which is used to trigger the image board.

For an operational test of the system, an array of LEDs was flashed with a 10 microsecond pulse and the image was grabbed (program 'grabber.c') and processed. (A 16 mm lens was attached to the camera.) Since the camera generates 30 frames per second, a pulse (T1) was sent to the digital input card 1/30 second before the LED flash. Upon receipt of T1, the image board is set up to grab the next frame. The camera will be at an arbitrary time in its image transfer cycle so the LED flash will likewise occur anywhere in the 1/30 second cycle time. However, the image is not uniformly dumped from the integrating pixels into shift registers every 1/30 second. In fact, half of the horizontal lines are shifted out every 1/30 second (say the odd lines) and the other half of the horizontal lines

are shifted out offset in time by 1/60 second. (Each of the groups is called a field, and the process is called interlacing. I believe it helps the eye and brain create a smoother image.) Figure 1 shows the frame grab timing. In case A both fields of the grabbed frame see the LED flash. In case B however, the second field does not see the LED flash because the flash occurred during the integration time of the previous frame's second field. To fix this up, after the frame is displayed on the monitor, the software runs through each horizontal line and copies the early field onto the late field (which may or may not be dark). This process degrades the vertical resolution a factor of two, but the system has oodles of spatial resolution so that's no problem.

Summarizing, for each LED flash a pulse is sent to the digital input board 1/30 second ahead of the LED flash. The frame is grabbed and displayed on the monitor and then the data are fudged to give a better looking image. The new image is then displayed on the monitor.

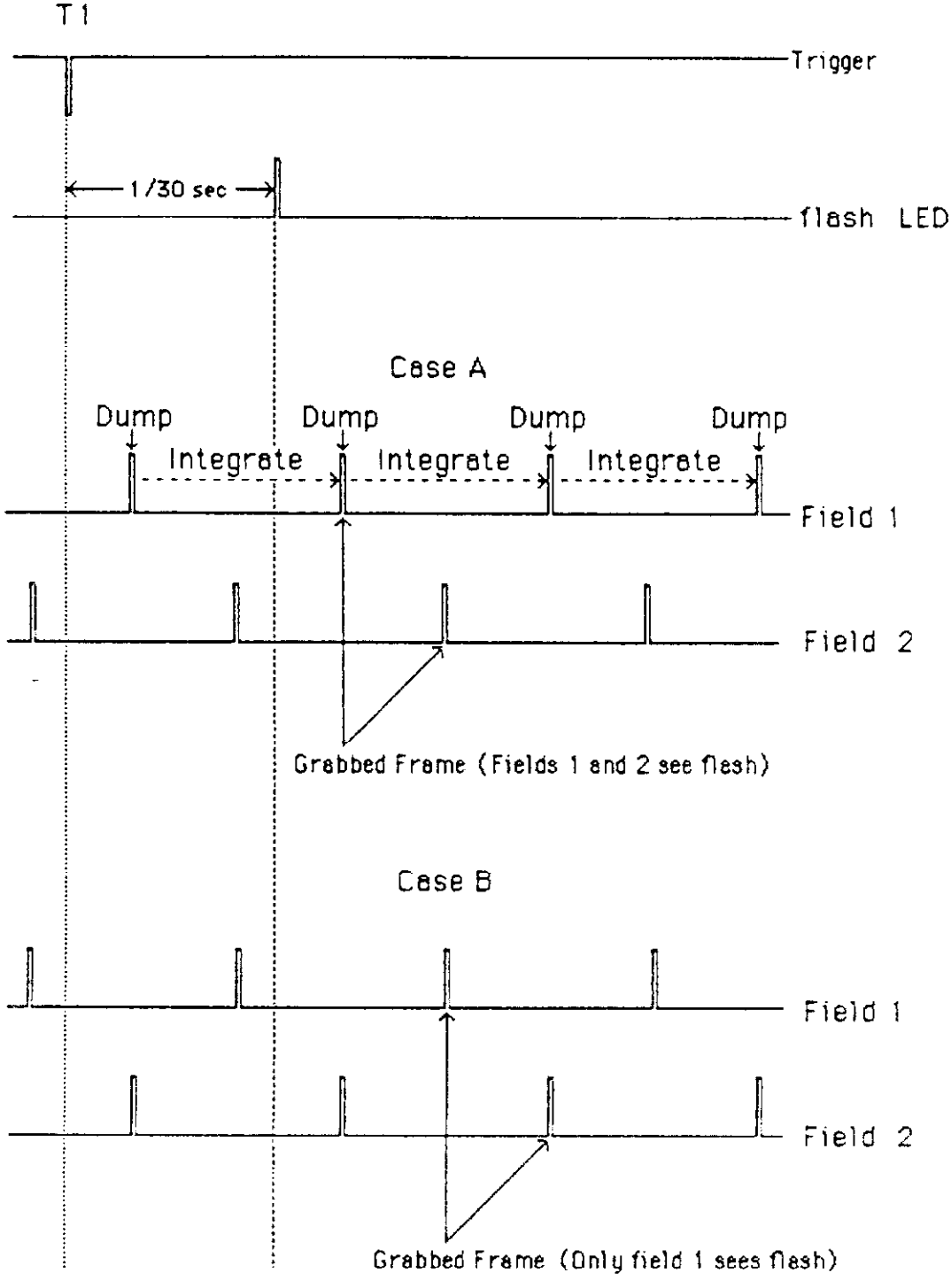
At this point some image processing gets done that demonstrates some of the potential of the system. A frame has 512 x 512 bytes of information, which is more than can be quickly transfered onto the experiment's data tapes using the present setup. So the program averages the pixels down to a 64 x 64 array which is then written on the PC's disk. This 4 kbyte array is plenty of data, enough to make a pretty clear driver's license sized picture. (Even 1 kbyte makes good postage stamp sized pictures.)

A couple of off-line type programs were made to massage the date some more. The first program (binbas.c) takes the 4 kbyte file and rewrites it in a format that 'basic' can use. After this is done, a simple basic program (mountn.bas) makes a 'mountain plot' of the data. Another 'basic' program (tshirt.bas) simulates the grayscale data with random dots so that a normal dot printer can make an image. (Another factor of four resolution

is lost in the process.) The source files, and output from these programs are included in Appendix 2.

Figure 1

Frame Grab Timing



The grabbed frame consists of the first complete field 1 and field 2 after the trigger.

Appendix I
Timing Jitter and Dead Time

The presence of the input card trigger signal is detected by a software loop which takes about 50 microseconds, hence to guarantee detection, the trigger pulse width must be greater than 50 microseconds.

Imprecise setting of the delay time between the input card trigger pulse and the LED flash will cause the wrong frame to be grabbed in some cases. If the delay is too long (look at Fig. 1, Case A, and stretch the '1/30 second' mark to the right until it passes another 'dump' mark) then there will be frames with the LED flash data only in Field 2. If the delay is too short (look at Fig. 1, Case B, and move the '1/30 second' mark to the left until it passes a 'dump' mark) there will be frames with no LED data in either field.

There is one more delay time which needs to be considered. Once the input trigger signal has been detected, a software command to grab a frame is executed. The time to generate the actual hardware frame grab signal from initiation of the software command has not been measured (call this time tg).

In principle, the total time delay from the beginning of the input card trigger and the LED flash whould be 1/30 second + 50 microseconds + tg.

The simple procedure used in 'grabber.c' always copies the early field onto the late field. If the timing is set as above, and the input trigger is detected early in the 50 microsecond software loop, then the delay is 'too long' by the remaining amount of the 50 microseconds. In this case, as described above, there will be frames with data only in field 2. After processing with 'grabber.c' the frame will be blank. The percentage of such frames will be (.5)x(.000050)x(30)=.075%. In practice, the delay was set to 1/30 second and 100 frames were grabbed with no misses, and the matter wasn't pursued further.

APPENDIX II

```
/*  ***** Program Grabber ***** */
/* This program grabs a frame upon receipt of a TTL BAR */
/* signal to the digital input card                     */
/* All of the functions whose names start with fg_ */
/* were supplied by Matrox with the purchase of */
/* the digitizing board.                          */
/* The program creates a file called riter3.bin */
#include <ctype.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include <stdlib.h>
main()
{
unsigned short fh1,fh2,buffer1[5000],ndx1,ndx2;
unsigned short ndx,pud,linefx;
int fh20,fh21;
char buffer[512];
short abyte;
int bits;
int del;
long addem;
int xpt,ypt,c1x,c2x,c1y,c2y;
bits=8;
del=4;
fh1=open("riter3.bin",O_RDWR|O_BINARY);
printf("%6d",fh1);
/* initialize the digitizer board */
fg_init(620);
fg_chan(2);
fg_sync(1);
fg_sbuf(0);
pool:
ndx1=0;ndx2=0;ndx=0;
loop:
/* Input to dim32 card */
abyte=inp(644);
/* Wait for a trigger on channel 0, which is base address */
/* 640. Channel 0 gives fe return (11111110) for the byte. */
if (abyte > 0xfe)
   goto loop;
if (abyte <0xfe)
   goto quit;
fg_snap(1);
fg_sbuf(1);
/*fix up line interlace shinola*/
linefx=0;
do{
   fg_rowr(linefx+1,0,buffer);
   fg_roww(linefx,0,buffer);
   linefx=linefx+2;
   }while (linefx < 511);
c1x=0;
c1y=1;
   do{
      c2y=c1y+bits;
      do{
         c2x=c1x+bits;
         /* box corners c1x,c1y,c2x,c2y*/
```

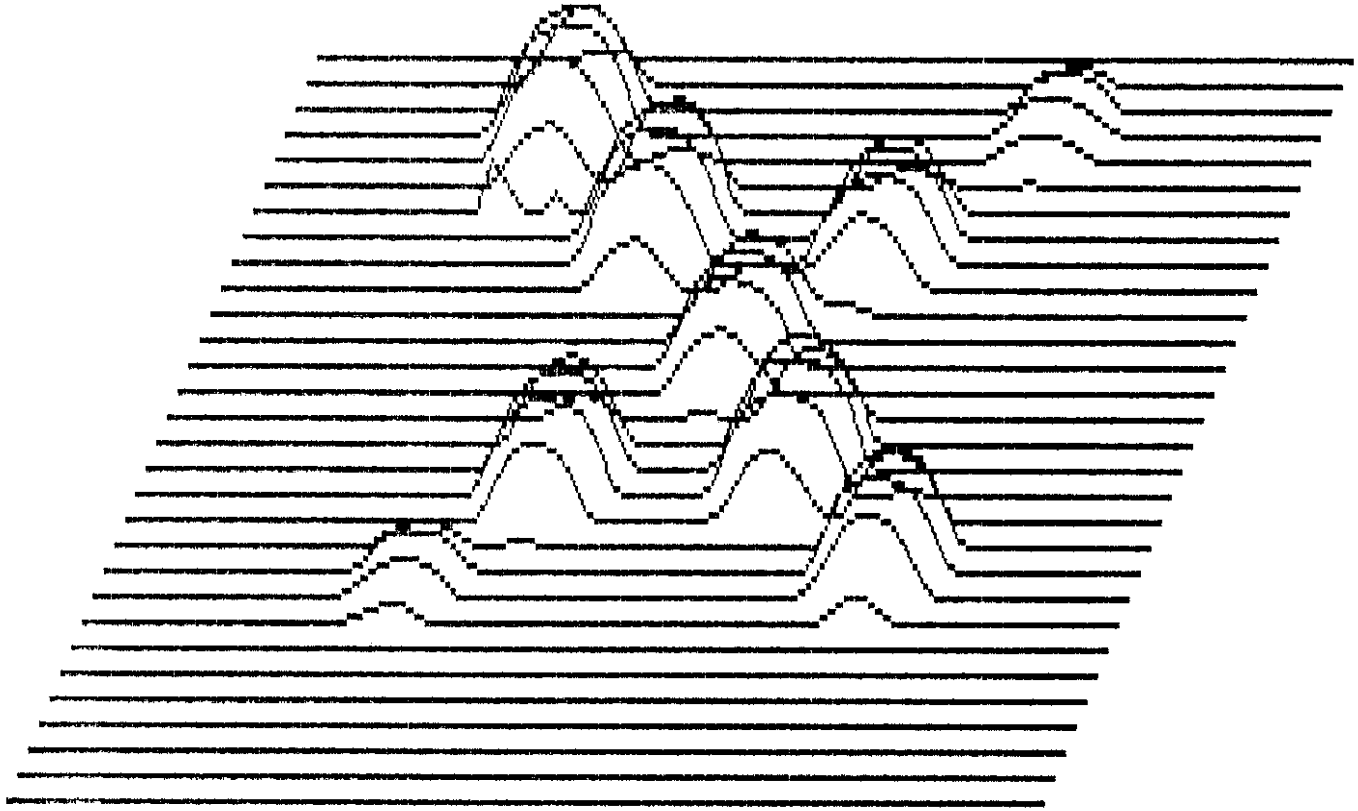
```
          addem=0;
          xpt=c1x;
          ypt=c1y;
             do{
                do{
                   addem=addem+fg_pixr(xpt,ypt);
                   xpt=xpt+del;
                  }while (xpt < c2x);
                xpt=c1x;
                ypt=ypt+del;
               }while (ypt < c2y);
          ypt=c1y;
          addem=addem/(bits*bits/(del*del));
         buffer1[ndx]=addem;
         ndx++;
          c1x=c1x+bits;
         }while (c1x < 512);
       c1x=0;
       c1y=c1y+bits;
     } while (c1y < (512-bits) );
do{
pud=((buffer1[ndx1+1])<<8)|buffer1[ndx1];
buffer1[ndx2]=pud;
ndx2++;ndx1++;ndx1++;
}while (ndx2 < 2048);
fh2=write(fh1,buffer1,4096);
printf("%6d",fh2);
if (abyte == 0xfe)
goto pool;
quit:
printf(" I quit ");
fg_exit();
}
```

```
/* ***** Program binbas ***** */
/* This program reads a file called riter3.bin and converts */
/* each byte to an integer in the format that basic can read */
/* The name of the written file is riter4.bin. */
#include <ctype.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include <stdlib.h>
char *yahoo;
int ndx9,bufind;
unsigned short fh1,fh4,bites,numb;
unsigned short fh3;
unsigned short ndx2,ndx3,ndx4;
unsigned short buffer2[12000];
unsigned short p0=0,p10=10,p13=13,p26=26;
unsigned short p1=1,p2,p3;
unsigned short len,fub1,fub2,fub3,fub4;
unsigned short p100,p101,p102,fh2,ndx;
unsigned short bite[2];
unsigned short left_over,mubuf1,mubuf2;
main()
{
ndx9=0;
left_over=0;
yahoo="cachunk";
printf(yahoo);
fh1=open("riter3.bin",O_RDONLY|O_BINARY);
fh2=open("riter4.bin",O_RDWR|O_BINARY);
printf("%8x",fh1);
bufind=p0;
bites=0;
do{
fh4=read(fh1,bite,1);
numb=bite[0]|p0;
p100=numb/100+48;
p101=(numb-100*(p100-48))/10+48;
p102=(numb-(p100-48)*100)-(p101-48)*10 +48;
if (p100 < 49)
   goto len2;
if (left_over =0){
     buffer2[bufind]=(p100<<8)|p101;
     bufind++;
     buffer2[bufind]=(p102<<8)|p13;
     bufind++;
     left_over=1;
     }
else{
     buffer2[bufind]=(p10<<8)|p100;
     bufind++;
     buffer2[bufind]=(p101<<8)|p102;
     bufind++;
     buffer2[bufind]=(p13<<8)|p10;
     bufind++;
     left_over=0;
     }
goto lupend;
len2:
```

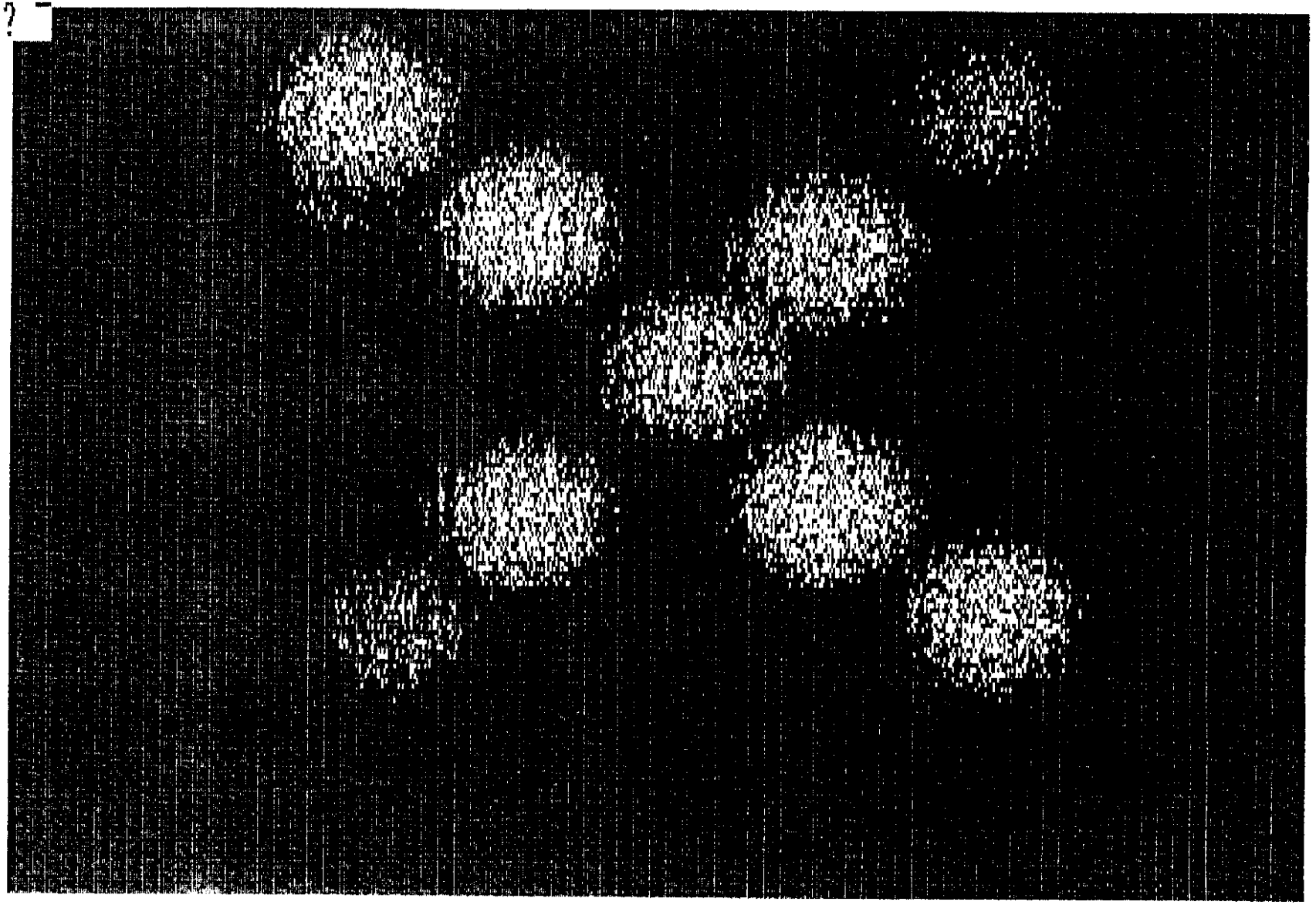
```
if (p101 <49)
goto len1;
if (left_over = 0){
      buffer2[bufind]=(p101<<8)|p102;
      bufind++;
      buffer2[bufind]=(p13<<8)|p10;
      bufind++;
      }
else{
      buffer2[bufind]=(p10<<8)|p101;
      bufind++;
      buffer2[bufind]=(p102<<8)|p13;
      bufind++;
      }
goto lupend;
len1:
if (left_over =0) {
       buffer2[bufind]=(p102<<8)|p13;
       bufind++;
       left_over=1;
      }
else{
      buffer2[bufind]=(p10<<8)|p102;
      bufind++;
      buffer2[bufind]=(p13<<8)|p10;
      bufind++;
      left_over=0;
      }
lupend:
bites++;
}while (bites < 4096);
printf("%6d",bufind);
if(left_over=1){
       buffer2[bufind]=(p13<<8)|p10;
       bufind++;
       buffer2[bufind]=(p26<<8)|p0;
       bufind++;
}
else{
       buffer2[bufind]=(p10<<8)|p26;
       bufind++;
}
do{
       mubuf1=(((buffer2[ndx9])&255)<<8);
       mubuf2=(buffer2[ndx9])>>8;
       buffer2[ndx9]=mubuf1|mubuf2;
       ndx9++;
}while (ndx9 < bufind);
/* now write file*/
fh3=write(fh2,buffer2,2*ndx9);
printf("%6x",fh3);
}
```

```
10 REM **** PROGRAM MOUNTN *****
20 REM This program reads a file called riter4.bin and makes a mountn
30 REM plot of the data. The assumption is made that the data corresponds
40 REM to a 64 by 64 array of 8 bit numbers. To make the plot easier
50 REM on the eye, only half of the vertical contours are plotted.
60 OPEN "i",#1,"riter4.bin"
70 SCREEN 2,0
80 CLS
90 ORIGX=200
100 ORIGY=30
110 FOR AY=2 TO 122 STEP 2
120 OLPIXL=0
130 FOR AX=1 TO 384 STEP 6
140 INPUT #1, PIXL
150 IF AY/4 <> INT(AY/4) THEN GOTO 190
160 PIXL=PIXL/10
170 LINE (ORIGX   +AX -AY,ORIGY  +AY-OLPIXL)-(ORIGX+6+AX-AY,ORIGY  +AY-PIXL)
180 OLPIXL=PIXL
190 NEXT AX
200 IF AY/4 <> INT(AY/4) THEN GOTO 220
210 LINE (ORIGX+AX-AY,ORIGY  +AY-PIXL)-(ORIGX+6+AX-AY,ORIGY  +AY)
220 NEXT AY
230 INPUT   XXX$
```

? MOUNTN.BAS OUTPUT (IBM PC MONITOR)



```
10 REM ***** PROGRAM TSHIRT *****
20 REM this program reads a file called riter4.bin and generates random
30 REM dots in a block 3 lines deep and ten columns wide. The number
40 REM dots is proportional to the 8 bit number read from riter4.bin.
50 REM There should be 4096 words in the file.
60 REM The blocks are arranged 64 horizontal and 60 vertical. (The last
70 REM few lines from the image digitizer are noise.)
80 OPEN "i",#1,"riter4.bin"
90 SCREEN 2,0
100 CLS
110 FOR B=1 TO 180 STEP 3
120 FOR A=5 TO 640 STEP 10
130 INPUT #1,PIXL
140 PIXL=255-PIXL
150 DENSITY=PIXL/255
160 FOR G=1 TO 3
170 FOR H=1 TO 10
180 K=RND
190 IF K > DENSITY THEN GOTO 210
200 PSET (A+H-1,B+G-1)
210 NEXT H
220 NEXT G
230 NEXT A
240 NEXT B
250 INPUT   XXX$
```



TSHIRT.BAS OUTPUT (IBM PC MONITOR)